Windows Driver -- 通过IDA逆向分析.sys

背景：因业务需要规划下一代PCIe SD host的Windows驱动，要支持Win11和以后的最新特性，因为现有的SD驱动是基于Storport-miniport架构，在Win11上有诸多限制严重影响业务，因此决定转型为WDF驱动。本文浅显分析Realtek的Win11 PCIe SD Card reader驱动是用什么架构，内部如何实现。

1. IDA反汇编工具

IDA能将二进制文件反汇编(disassemble)成为汇编代码，还支持将汇编代码进一步显示成C代码(decompile)。

下载IDA free版本就够用

https://hex-rays.com/products/ida/support/%20download_freeware%20/

2. IDA分析驱动.sys文件

2.1. IDA常用快捷键

F5：汇编代码转C代码显示（IDA称为伪代码，因为不是纯C）

Shift + F12：显示所有符号的字符串。可以全览所有函数，弄清用的什么技术栈

x：查看函数和变量的交叉引用，即被谁调用

esc：返回上个页面位置

2.2. 驱动分析示例

2.2.1. RtsPer.sys下载

https://www.driverscloud.com/en/services/GetInformationDriver/75616-0/realtek-cardreader-win10-win11-1002262121361zip

2.2.2. INF分析

```
[Version]
Signature="$Windows NT$"
Class=MTD
ClassGuid = {4d36e970-e325-11ce-bfc1-08002be10318}
Provider=%RTS%
CatalogFile = RtsPer64.cat
DriverVer=11/14/2022,10.0.22621.21361

... 以下以Rts5227CR为例

[DestinationDirs]
CopyFilesSYS = 12     ; should it be 10 to take care of 98 stuff
CopyFilesDLL = 11     ; %SystemRoot%\system or system32 - 98 or Win2000
CopyFilesDLL64 = 10,SysWOW64

[Manufacturer]
%VENDOR%=Vendor, NTamd64

[Vendor.NTamd64]
```

```
%Rts5227CR%=RTS5264.Inst, PCI\VEN_10EC&DEV_5264&CC_FF00

[RTS5264.Inst.ntamd64]
CopyFiles = CopyFilesSYS, CopyFilesDLL64

[RTS5264.Inst.NTamd64.HW]
AddReg=MsiEnable_addreg

[RTS5264.Inst.ntamd64.Services]
AddService = RTSPER, 0x00000002, RTS5264_Service_Inst

[RTS5264_Service_Inst]
DisplayName    = %Rts5227PER%
ServiceType    = %SERVICE_KERNEL_DRIVER%
StartType      = %SERVICE_DEMAND_START%
ErrorControl   = %SERVICE_ERROR_IGNORE%
ServiceBinary  = %12%\RtsPer.sys
AddReg         = RTS5264.AddReg

[RTS5264.AddReg]
HKR,"RTS5264","MSIEnable",0x10001,1
HKR,"RTS5264","FirstLoad",0x10001,1
HKR,"RTS5264","NonRemovable",0x10001,1
HKR,"RTS5264","SupportPoFx",0x10001,1
HKR,"Parameters","DmaRemappingCompatible",0x10001,1

[Strings]
;Localizable Strings needed for HBA naming in Windows UI
;*****************************************
;Non-Localizable strings
RTS = "Realtek Semiconductor Corp."
VENDOR         = "Realtek Semiconductor Corp."
Rts5227CR      = "Realtek PCIE CardReader"
Rts5227PER      = "Realtek PCIE Card Reader - PER"
DiskDesc = "Realtek PCIE Card Reader Source Disk"
DriverVersion = "10.0.22621.21361"
SERVICE_ASSOCSERVICE = 0x00000002
SERVICE_BOOT_START     = 0x0
SERVICE_SYSTEM_START   = 0x1
SERVICE_AUTO_START     = 0x2
SERVICE_DEMAND_START   = 0x3
SERVICE_DISABLED       = 0x4
SERVICE_KERNEL_DRIVER  = 0x1
SERVICE_ERROR_IGNORE   = 0x0
SERVICE_ERROR_NORMAL   = 0x1
SERVICE_ERROR_SEVERE   = 0x2
SERVICE_ERROR_CRITICAL = 0x3
REG_EXPAND_SZ          = 0x00020000
REG_DWORD              = 0x00010001
REG_MULTI_SZ           = 0x00010000
REG_BINARY             = 0x00000001
REG_SZ                 = 0x00000000
```

设备类型是MTD：**Memory Technology Driver**

https://learn.microsoft.com/en-us/windows-hardware/drivers/install/system-defined-device-setup-classes-available-to-vendors

从INF可以推测：

（1）这是SD host设备的驱动，直连PCIe接口（没通过USB），作用是SD card的控制器。

（2）没有用WDF(KMDF)框架，因为KMDF的INF一般定义KmdfService字段，以上INF没有定义。

（3）结合Windows Driver Sample，MTD类属于SD BUS/Device的设备驱动，但微软的SD框架不支持SD BUS只支持SD Device，因此该驱动应该是用WDM写的SD BUS驱动，不是依赖于微软的SD BUS框架。

INF详细WDK 文档：https://learn.microsoft.com/en-us/windows-hardware/drivers/install/looking-at-an-inf-file

2.2.3. .sys分析

1. IDA打开.sys (有pdb文件更好)，找到DriverEntry入口

2. F5显示成C伪代码，可以双击函数跳转

3. 详细分析一下Driver Entry做了什么：

```
__int64 __fastcall sub_1400DEE88(_QWORD *a1) //DriverEntry主功能在这里实现，所有叫sub_xxx函数都是没有符号表解析不出名字的函数，看函数体即可.
{
  __int64 CurrentThreadId; // rax
  __int64 v3; // rax
  __int64 v4; // rcx
  __int64 v5; // rax
  __int64 v7; // [rsp+30h] [rbp-148h]
  struct _OSVERSIONINFOW VersionInformation; // [rsp+40h] [rbp-138h] BYREF

  CurrentThreadId = PsGetCurrentThreadId(); //获取当前线程ID
  sub_1400DE608( //根据函数体，这里只是个打印函数，打印当前时间和线程ID.
    2LL,
    "%I64d (%d) %s : -> DriverEntry built on %s at %s \n",
    (MEMORY[0xFFFFF78000000008] - qword_140140278) / 0x2710uLL,
    CurrentThreadId,
    "DriverEntry",
    "Nov 14 2022",
    "15:12:36");
  v3 = PsGetCurrentThreadId();
  sub_1400DE608(
    2LL,
    "%I64d (%d) %s : -> DriverEntry Driver version : %s \n",
    (MEMORY[0xFFFFF78000000008] - qword_140140278) / 0x2710uLL,
    v3,
    "DriverEntry",
    "10.0.22621.21361");
  VersionInformation.dwOSVersionInfoSize = 276;
  if ( RtlGetVersion(&VersionInformation) >= 0 //获取操作系统版本
    && (VersionInformation.dwMajorVersion > 6
     || VersionInformation.dwMajorVersion == 6 &&
VersionInformation.dwMinorVersion >= 2) )
  {
    dword_140140110 = 512;
    dword_140140114 = 0x40000000;
```

```
  }
  qword_140140278 = MEMORY[0xFFFFF78000000008];
  sub_1400DEC80();
  sub_1400109EC();
  a1[28] = sub_140003340; //sub_xxx都是函数，所以这里是注册很多回调，根据WDM开发一般是PNP回调
  a1[29] = sub_1400057A0;
  a1[36] = sub_1400EFA30;
  a1[13] = sub_1400E0670;
  v4 = a1[6];
  a1[41] = sub_1400E5480;
  a1[14] = sub_140003000;
  a1[16] = sub_140002DC0;
  a1[32] = sub_140002910;
  a1[30] = sub_140008B40;
  a1[37] = sub_1401071C0;
  a1[18] = sub_140006EA0;
  a1[17] = sub_140006EA0;
  *(_QWORD *)(v4 + 8) = sub_1400DF0F0;
  sub_1400011E4();
  sub_1400E0188();
  v5 = PsGetCurrentThreadId();
  LODWORD(v7) = 0;
  sub_1400DE608(
    0x2000LL,
    "%I64d (%d) %s : <- %s, ret = 0x%x\n",
    (MEMORY[0xFFFFF78000000008] - qword_140140278) / 0x2710uLL,
    v5,
    "DriverEntry",
    "DriverEntry",
    v7);
  return 0LL;
}
```

具体看一下注册的回调函数的内容：在函数上按x找到所有引用，再F5查看C伪代码

```
__int64 __fastcall sub_1400057A0(__int64 a1, __int64 a2)
{
  __int64 v2; // rdi
  __int64 CurrentThreadId; // rax
  unsigned int v5; // ebx
  __int64 v6; // rax
  int v8; // [rsp+30h] [rbp-18h]

  v2 = *(_QWORD *)(a1 + 64);
  if ( *(_BYTE *)v2 != 1 )
    return sub_1400058A0();
  CurrentThreadId = PsGetCurrentThreadId();
  sub_1400DE608(
    0x2000,
    "%I64d (%d) %s : -> %s\n",
    (MEMORY[0xFFFFF78000000008] - qword_140140278) / 0x2710uLL,
    CurrentThreadId,
    "rts_internalctrl",
    "rts_internalctrl");
  ++*(_BYTE *)(a2 + 67);
  *(_QWORD *)(a2 + 184) += 72LL;
```

```
  //注意这个IofCallDriver，用于转发IRP给设备的driver function.
  //可以推测DriverEntry注册的那些回调函数就是注册PNP请求列表对应的处理函数
  //这里仅转发，真正的处理逻辑还在下层函数
  v5 = IofCallDriver(*(_QWORD *)(v2 + 16), a2);
  v6 = PsGetCurrentThreadId();
  v8 = v5;
  sub_1400DE608(
    0x2000,
    "%I64d (%d) %s : <- %s, ret = 0x%x\n",
    (MEMORY[0xFFFFF78000000008] - qword_140140278) / 0x2710uLL,
    v6,
    "rts_internalctrl",
    "rts_internalctrl",
    v8);
  return v5;
}
```

PNP的回调函数参考：https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/dispatchpnp-routines

4. 全览.sys有哪些函数符号

   用shift + F12打开strings页面，ctrl+F 搜索关键词，选中结果后删除搜索框去浏览上下文。

以DriverEntry为例，符号字符上下文如下，符号的地址分布是连续的，不考虑跳转可视为调用顺序。

```
.text:000000014012A550  0000000C    C   DriverEntry //入口
.text:000000014012A560  00000033    C   %I64d (%d) %s : -> DriverEntry built on
%s at %s \n
.text:000000014012A5A0  00000035    C   %I64d (%d) %s : -> DriverEntry Driver
version : %s \n
.text:000000014012A5E0  0000000B    C   rts_unload
.text:000000014012A5F0  00000031    C   %I64d (%d) %s : -> Driver Unload,
version : %s \n
.text:000000014012A630  00000031    C   %I64d (%d) %s : <- Driver Unload,
version : %s \n
.text:000000014012A670  0000000E    C   rts_adddevice
.text:000000014012A6C0  00000031    C   %I64d (%d) %s : Failed to create device
object \n
.text:000000014012A700  00000031    C   %I64d (%d) %s : fdx is 0x%p, PAGE_SIZE
is 0x%x \n
.text:000000014012A740  00000035    C   %I64d (%d) %s :
IoAttachDeviceToDeviceStack failed \n

//操作系统判断，为了后面差异化配置

.text:000000014012A780  0000001E    C   %I64d (%d) %s : OS is Win10 \n
.text:000000014012A7A0  00000020    C   %I64d (%d) %s : OS is WinBlue \n
.text:000000014012A7C0  0000001D    C   %I64d (%d) %s : OS is Win8 \n
.text:000000014012A7E0  0000001D    C   %I64d (%d) %s : OS is Win7 \n
.text:000000014012A800  00000024    C   %I64d (%d) %s : OS is Server 2008 \n
.text:000000014012A830  0000001E    C   %I64d (%d) %s : OS is VISTA \n
.text:000000014012A850  00000024    C   %I64d (%d) %s : OS is Server 2003 \n
.text:000000014012A880  0000001E    C   %I64d (%d) %s : OS is WinXp \n
.text:000000014012A8A0  0000001E    C   %I64d (%d) %s : OS is Win2k \n
.text:000000014012A8C0  00000023    C   %I64d (%d) %s : OS is NotDefined \n
```

```
.text:000000014012A8F0  00000001    C   %I64d (%d) %s : OS is 64bit \n
.text:000000014012A910  0000003A    C   %I64d (%d) %s :
rts_pcie_init_bus_interface failed (%x) \n
.text:000000014012A950  00000035    C   %I64d (%d) %s : rts_pcie_get_dev_info
failed: 0x%x \n
.text:000000014012A990  0000003B    C   %I64d (%d) %s :
IoRegisterDeviceInterface failed with %x \n
.text:000000014012A9D0  00000032    C   %I64d (%d) %s : GetMcfgEntryFromAuxKlib
success \n
.text:000000014012AA10  00000031    C   %I64d (%d) %s : GetMcfgEntryFromAuxKlib
failed \n
.text:000000014012AA50  0000002E    C   %I64d (%d) %s : GetMcfgEntryFromReg
success \n
.text:000000014012AA80  0000002D    C   %I64d (%d) %s : GetMcfgEntryFromReg
failed \n
.text:000000014012AAB0  00000027    C   %I64d (%d) %s : host_cfg_disable: %d \n
.text:000000014012AAE0  00000034    C   %I64d (%d) %s : bMcfgEntry %d, BaseAddr
is 0x%llx \n

//初始化DPC，PDO；Rts自定义的PNP/POFX回调函数也初始化（估计是绑定到函数指针数组）

.text:000000014012AB20  00000041    C   %I64d (%d) %s : initialize the DPC
NoSSDpcWorkItemPendingEvent \n
.text:000000014012AB70  00000039    C   %I64d (%d) %s :
IoRegisterShutdownNotification success \n
.text:000000014012ABB0  0000003B    C   %I64d (%d) %s :
IoRegisterShutdownNotification fail 0x%x \n
.text:000000014012ABF0  00000015    C   rts_create_child_pdo
.text:000000014012AC10  0000005A    C   %I64d (%d) %s : Create Pdo %i
successfully, status is 0x%x, Child->ReferenceCount is %i \n
.text:000000014012AC70  00000035    C   %I64d (%d) %s : Create Pdo %i fail with
status 0x%x\n
.text:000000014012ACE0  00000017    C   rts_init_pofx_routines
.text:000000014012AD00  0000002D    C   %I64d (%d) %s :
pPoFxActivateComponent=0x%p\n
.text:000000014012AD60  00000029    C   %I64d (%d) %s :
pPoFxIdleComponent=0x%p\n
.text:000000014012ADC0  0000002C    C   %I64d (%d) %s :
pPoFxSetComponentWake=0x%p\n
.text:000000014012AE20  0000002D    C   %I64d (%d) %s :
pPoFxCompleteIdleState=0x%p\n
.text:000000014012AE90  00000031    C   %I64d (%d) %s :
pPoFxCompleteIdleCondition=0x%p\n
.text:000000014012AF10  00000031    C   %I64d (%d) %s :
pPoFxReportDevicePoweredOn=0x%p\n
.text:000000014012AFA0  0000003A    C   %I64d (%d) %s :
pPoFxCompleteDevicePowerNotRequired=0x%p\n
.text:000000014012B010  0000002A    C   %I64d (%d) %s :
pPoFxRegisterDevice=0x%p\n
.text:000000014012B070  0000002C    C   %I64d (%d) %s :
pPoFxUnregisterDevice=0x%p\n
.text:000000014012B0E0  00000036    C   %I64d (%d) %s :
pPoFxStartDevicePowerManagement=0x%p\n
.text:000000014012B160  00000035    C   %I64d (%d) %s :
pPoFxCompleteDirectedPowerDown=0x%p\n

//从注册表拿到用户自定义的功能配置信息
```

```
.text:000000014012B1B0  00000014    C   GetMcfgEntryFromReg
.text:000000014012B220  00000028    C   %s : Get SubKey %ws, Update Key to %ws\n
.text:000000014012B250  0000003F    C   %I64d (%d) %s : EnumOneSubValue return
%d, pMcfgSdtTabke 0x%x\n
.text:000000014012B290  00000023    C   %I64d (%d) %s : NULL == pMcfgAddr\n
.text:000000014012B2C0  00000018    C   GetMcfgEntryFromAuxKlib
.text:000000014012B2E0  00000030    C   %I64d (%d) %s : Enum firmware table
return %#x\n
.text:000000014012B310  00000041    C   %I64d (%d) %s :
AuxKlibEnumerateSystemFirmwareTables return %#x\n
.text:000000014012B360  00000023    C   %I64d (%d) %s : cannot find MCFG \n
.text:000000014012B390  0000002C    C   %I64d (%d) %s : Get MCFG Table as
follow: \n
.text:000000014012B3C0  00000031    C   %I64d (%d) %s : Allocate for MCFG table
failed \n
.text:000000014012B400  0000000E    C   EnumOneSubKey
.text:000000014012B410  00000029    C   %s : Open register key %ws failed,
0x%x\n
.text:000000014012B440  00000010    C   EnumOneSubValue
.text:000000014012B450  00000026    C   %I64d (%d) %s : Allocate pfi failed \n
.text:000000014012B480  00000027    C   %I64d (%d) %s : Allocate pvfi failed \n
.text:000000014012B4B0  00000024    C   %I64d (%d) %s : DataLength is 0x%x\n
.text:000000014012B4E0  0000000E    C   ParseSdtTable
.text:000000014012B4F0  0000002F    C   %I64d (%d) %s : Check physical address
%#llx \n
.text:000000014012B520  0000002D    C   %I64d (%d) %s : Check physical address
%#x \n

//从PCIe bridge拿到SD host设备信息，包括能力寄存器，电源ASPM等

.text:000000014012B550  00000018    C   rts_get_pci_bridge_info
.text:000000014012B570  00000048    C   %I64d (%d) %s : Single Function Device:
bus = %#x, dev = %#x, func=%#x\n
.text:000000014012B5C0  0000003C    C   %I64d (%d) %s : Find Device(%X:%X)
bus=%d dev=%d, func=%d\n
.text:000000014012B600  00000031    C   %I64d (%d) %s : Save host configure
space 0x%p \n
.text:000000014012B640  00000033    C   %I64d (%d) %s : Cannot Find PciBridge
for Device \n
.text:000000014012B680  0000001B    C   rts_get_dev_link_ctl_field
.text:000000014012B6A0  00000039    C   %I64d (%d) %s : Get PCI_COMMON_CONFIG
fail, ulResult=%d\n
.text:000000014012B6E0  00000033    C   %I64d (%d) %s : Get linkCtrlReg fail,
ulResult=%d\n
.text:000000014012B720  00000022    C   %I64d (%d) %s : linkCtrlReg 0x%x\n
.text:000000014012B750  00000022    C   rts_get_bridge_link_control_field
.text:000000014012B780  0000002E    C   %I64d (%d) %s : fail to find PCIe
Capability\n
.text:000000014012B7B0  0000003B    C   %I64d (%d) %s : CapabilityOffset -
Config from MMCFG 0x%x\n
.text:000000014012B7F0  00000038    C   %I64d (%d) %s : CapabilityOffset -
Config from IO 0x%x\n
.text:000000014012B830  00000038    C   %I64d (%d) %s : CapabilityHdr - Config
from MMCFG 0x%x\n
.text:000000014012B870  00000035    C   %I64d (%d) %s : CapabilityHdr - Config
from IO 0x%x\n
.text:000000014012B8B0  00000036    C   %I64d (%d) %s : LinkCtrlReg - Config
from MMCFG 0x%x\n
```

```
.text:000000014012B8F0  00000033    C   %I64d (%d) %s : LinkCtrlReg - Config
from IO 0x%x\n
.text:000000014012B930  00000048    C   %I64d (%d) %s : pciBridgePCIeHdrOffset
0x%x, pciBridgeLinkCtrlReg 0x%x\n
.text:000000014012B980  0000002D    C   %I64d (%d) %s : Cannot Find PCIe
Capability\n
.text:000000014012B9B0  00000038    C   %I64d (%d) %s : cannot find the Bus of
PCI,do nothing \n
.text:000000014012B9F0  00000023    C   %I64d (%d) %s : MapPhyMem failed \n
.text:000000014012BA20  0000005F    C   %I64d (%d) %s : PciBridge BusNumber[%x],
DevNumbe[%x], FuncNumber[%x], Write reg[0x%x] = 0x%x\n
.text:000000014012BA80  00000016    C   rts_disable_host_aspm
.text:000000014012BAA0  0000004A    C   %I64d (%d) %s : recognize the Bus of
PCI(Bridge) as UNKNOWN, do nothing \n
.text:000000014012BAF0  0000001F    C   %I64d (%d) %s : PhyAddr 0x%x \n
.text:000000014012BB10  0000002A    C   %I64d (%d) %s : Offset 0x%x, Value 0x%x
\n
.text:000000014012BB40  00000012    C   rts_set_host_aspm
.text:000000014012BB60  0000002A    C   %I64d (%d) %s : Offset 0x%x, value 0x%x
\n
.text:000000014012BB90  00000012    C   rts_get_host_aspm
.text:000000014012BBB0  0000001D    C   rts_pci_find_host_capability
.text:000000014012BBD0  00000019    C   cr_read_host_config_byte
.text:000000014012BBF0  0000001A    C   cr_write_host_config_byte
.text:000000014012BC10  0000002F    C   %I64d (%d) %s : Write configure through
MMIO \n
.text:000000014012BC40  00000007    C   UNKNOW

//以下是PNP的回调函数的注册（函数指针绑定），具体函数体实现在rts_pnp_fdo

.text:000000014012BC70  00000011    C   DispatchPnP_Fdo
.text:000000014012BC90  0000000C    C   rts_pnp_fdo
.text:000000014012BCA0  0000001E    C   %I64d (%d) %s : -> %s %s %s \n
.text:000000014012BCC0  00000032    C   %I64d (%d) %s : rts_pnp_fdo: fdx
DeviceState %i \n
.text:000000014012BD00  00000052    C   %I64d (%d) %s : IRP_MN_REMOVE_DEVICE,
NotStarted == fdx->DeviceState, do nothing\n
.text:000000014012BD60  0000002D    C   %I64d (%d) %s : Removed == fdx-
>DeviceState\n
.text:000000014012BD90  00000028    C   %I64d (%d) %s : call rts_ss_cancel_ss \n
.text:000000014012BDD0  0000004F    C   %I64d (%d) %s : MSI not
enable,IRP_MN_FILTER_RESOURCE_REQUIREMENTS to default\n
.text:000000014012BE20  00000035    C   %I64d (%d) %s : Unprocessed pnp,to
default process \n
.text:000000014012BE60  00000011    C   DispatchPnP_Pdo
.text:000000014012BE80  0000000C    C   rts_pnp_pdo
.text:000000014012BE90  00000061    C   %I64d (%d) %s : NULL == Fdo, not
IRP_MN_REMOVE_DEVICE,so return fail with STATUS_DELETE_PENDING\n
.text:000000014012BF00  00000037    C   %I64d (%d) %s : Removed == fdx-
>DeviceState, so quit \n
.text:000000014012BF40  00000027    C   %I64d (%d) %s : NULL == fdx, so quit \n
.text:000000014012BF70  0000001F    C   %I64d (%d) %s : BusRelations \n
.text:000000014012BF90  00000024    C   %I64d (%d) %s : EjectionRelations \n
.text:000000014012BFC0  00000021    C   %I64d (%d) %s : PowerRelations \n
.text:000000014012BFF0  00000023    C   %I64d (%d) %s : RemovalRelations \n
.text:000000014012C020  00000027    C   %I64d (%d) %s : TargetDeviceRelation \n
```

```
.text:000000014012C050  00000097    C   %I64d (%d) %s : deviceCapabilities->Removable is %i,deviceCapabilities->SurpriseRemovalOK is %i,deviceCapabilities->UniqueID is %i, ntStatus is 0x%x \n
.text:000000014012C0F0  00000017    C   rts_tr_pcie_option_set
.text:000000014012C110  00000024    C   %I64d (%d) %s : sd_capability=%#x \n
.text:000000014012C140  00000023    C   %I64d (%d) %s : card_spt_map=%#x \n
.text:000000014012C170  0000002D    C   %I64d (%d) %s : cr->option.dev_flags = %#x \n
.text:000000014012C1A0  0000002F    C   %I64d (%d) %s : cr->option.patch_flags = %#x \n

//以下是设备资源分配（xxx_alloc）和硬件寄存器值初始化（bios_setting/init_hw），由于是PCIe的SD host设备，主要分配SD host设备空间到PCIe bar地址

.text:000000014012C1D0  00000045    C   %I64d (%d) %s : DriverFirstLoad, set delink_delay_max_cnt to %d ms \n
.text:000000014012C220  00000026    C   %I64d (%d) %s : Clar firstload flag \n
.text:000000014012C250  00000036    C   %I64d (%d) %s : fdx->cr->option.remote_wakeup_en=%#x\n
.text:000000014012C290  00000038    C   %I64d (%d) %s : fdx->CurrentPara->remote_wakeup_en=%#x\n
.text:000000014012C2D0  00000036    C   %I64d (%d) %s : fdx->cr->option.host_cfg_disable=%#x\n
.text:000000014012C310  00000020    C   rts_tr_pcie_backup_bios_setting
.text:000000014012C330  00000015    C   rts_cr_bind_together
.text:000000014012C350  0000002F    C   %I64d (%d) %s : cr=%p, cm =%p, tr=%p, scsi=%p\n
.text:000000014012C380  00000033    C   %I64d (%d) %s : cr->cm=%p, cr->tr=%p, cr->scsi=%p\n
.text:000000014012C3C0  00000033    C   %I64d (%d) %s : cm->cr=%p, cm->tr=%p, scsi->cr=%p\n
.text:000000014012C400  0000000B    C   scsi_alloc
.text:000000014012C410  0000002E    C   %I64d (%d) %s : Unable to allocate the scsi \n
.text:000000014012C440  0000000D    C   scsi_release
.text:000000014012C450  0000001B    C   rts_option_set_bef_init_hw
.text:000000014012C470  00000059    C   %I64d (%d) %s : read config addr 0x0E to judge multi function fail, bytesread(%i) != 1 \n
.text:000000014012C4D0  00000036    C   %I64d (%d) %s : read config addr 0x0E success(0x%x) \n
.text:000000014012C510  00000006    C   multi
.text:000000014012C520  00000007    C   single
.text:000000014012C530  00000029    C   %I64d (%d) %s : Device is %s-functioned\n
.text:000000014012C560  0000002F    C   %I64d (%d) %s : fdx->cr->option.adma_mode %d \n
.text:000000014012C590  0000001D    C   rts_option_set_after_init_hw
.text:000000014012C5B0  0000002C    C   %I64d (%d) %s : option.cq_rand_enable = %d\n
.text:000000014012C5E0  0000002B    C   %I64d (%d) %s : option.cq_seq_enable = %d\n
.text:000000014012C610  00000030    C   %I64d (%d) %s : option.cq_ban_card_enable = %d\n
.text:000000014012C640  00000020    C   Realtek PCIE Card Reader Driver
.text:000000014012C660  00000011    C   rts_cr_init_comm
.text:000000014012C680  0000001E    C   %I64d (%d) %s : %s detected \n
.text:000000014012C6A0  00000023    C   %I64d (%d) %s : option->ss_en %d \n
.text:000000014012C6D0  00000013    C   rts_cr_uninit_comm
```

```
//以下是电源管理POFX的回调函数的注册（函数指针绑定），具体实现在rts_pofx/dfx

.text:000000014012C6F0  00000018    C   ActiveConditionCallback
.text:000000014012C710  0000003A    C   %I64d (%d) %s : ===><=== -> PoFx
ActiveConditionCallback\n
.text:000000014012C750  0000003A    C   %I64d (%d) %s : ===><=== <- PoFx
ActiveConditionCallback\n

.text:000000014012C790  00000016    C   IdleConditionCallback
.text:000000014012C7B0  00000038    C   %I64d (%d) %s : ===><=== -> PoFx
IdleConditionCallback\n
.text:000000014012C7F0  00000038    C   %I64d (%d) %s : ===><=== <- PoFx
IdleConditionCallback\n
.text:000000014012C830  00000012    C   IdleStateCallback
.text:000000014012C850  0000003E    C   %I64d (%d) %s : ===><=== -> PoFx
IdleStateCallback, State=%d\n
.text:000000014012C890  00000034    C   %I64d (%d) %s : ===><=== <- PoFx
IdleStateCallback\n

.text:000000014012C8D0  0000001C    C   DevicePowerRequiredCallback
.text:000000014012C8F0  00000037    C   %I64d (%d) %s : ===> PoFx
DevicePowerRequiredCallback\n
.text:000000014012C930  00000041    C   %I64d (%d) %s : PoFx
DeviePowerRequiredCallback:queue work item\n
.text:000000014012C980  00000053    C   %I64d (%d) %s : PoFx
DeviePowerRequiredCallback:Cannot alloc memory for work item\n
.text:000000014012C9E0  0000002A    C   %I64d (%d) %s :  fdx->PoFxActive = TRUE
\n
.text:000000014012CA10  0000003E    C   %I64d (%d) %s : ===><=== <- PoFx
DevicePowerRequiredCallback\n

.text:000000014012CA50  0000001F    C   DevicePowerNotRequiredCallback
.text:000000014012CA70  0000003A    C   %I64d (%d) %s : ===> PoFx
DevicePowerNotRequiredCallback\n
.text:000000014012CAB0  00000043    C   %I64d (%d) %s : ===><=== PoFx
pPoFxCompleteDevicePowerNotRequired\n
.text:000000014012CB00  0000002B    C   %I64d (%d) %s :  fdx->PoFxActive = FALSE
\n
.text:000000014012CB30  0000003A    C   %I64d (%d) %s : <=== PoFx
DevicePowerNotRequiredCallback\n

//以下是电源管理POFX的回调函数体实现，rts_xxx_pofx

.text:000000014012CB70  00000025    C   rts_dev_pwr_completion_for_DFx_child
.text:000000014012CBA0  0000004A    C   %I64d (%d) %s : powerContext-
>DeviceObject is 0x%p, DeviceObject is 0x%p\n
.text:000000014012CBF0  00000027    C   %I64d (%d) %s : IoStatus->Status=%#x \n

.text:000000014012CC20  0000001F    C   rts_dev_pwr_completion_for_DFx
.text:000000014012CC40  00000079    C   %I64d (%d) %s : powerContext-
>DeviceObject is 0x%p, DeviceObject is 0x%p,deviceExtension->PhysicalDeviceObject
is 0x%p \n
.text:000000014012CCC0  00000030    C   %I64d (%d) %s : For power up, queue work
item \n
.text:000000014012CCF0  00000041    C   %I64d (%d) %s : For power up, cannot
alloc memory for work item\n
```

```
.text:000000014012CD40  00000038    C   %I64d (%d) %s : ===><===
PoFxCompleteDirectedPowerDown\n
.text:000000014012CD80  0000002D    C   %I64d (%d) %s : Set Power Failed, resume
IO\n

.text:000000014012CDB0  00000018    C   DirectedPowerUpCallback
.text:000000014012CDD0  0000003A    C   %I64d (%d) %s : ===><=== -> PoFx
DirectedPowerUpCallback\n
.text:000000014012CE10  0000003A    C   %I64d (%d) %s : Failed to alloc memory
for powerContext \n
.text:000000014012CE50  0000003E    C   %I64d (%d) %s : DirectedPowerUpCallback
SetPower to D0 fail \n

.text:000000014012CE90  0000001A    C   DirectedPowerDownCallback
.text:000000014012CEB0  0000003C    C   %I64d (%d) %s : ===><=== -> PoFx
DirectedPowerDownCallback\n
.text:000000014012CEF0  0000002E    C   %I64d (%d) %s : Set enter_rtd3 to 1 for
DFx \n
.text:000000014012CF20  00000040    C   %I64d (%d) %s :
DirectedPowerDownCallback SetPower to D3 fail \n

.text:000000014012CF60  00000012    C   rts_register_pofx
.text:000000014012CF80  00000035    C   %I64d (%d) %s : supportPoFx=0, do not
register PoFx\n
.text:000000014012CFC0  00000033    C   %I64d (%d) %s : ===><=== PoFx already
registered!\n
.text:000000014012D000  00000042    C   %I64d (%d) %s : ===><=== Invalid PoFx
routines, pls check again!\n
.text:000000014012D050  00000037    C   %I64d (%d) %s : ===><=== OS Ver: size
%d, %d.%d.%d.%d\n
.text:000000014012D090  00000033    C   %I64d (%d) %s : ===><=== Register
PO_FX_DEVICE_V3\n
.text:000000014012D0D0  00000033    C   %I64d (%d) %s : ===><=== Register
PO_FX_DEVICE_V2\n
.text:000000014012D110  00000038    C   %I64d (%d) %s : ===><===
PoFxRegisterDevice return %#x\n

.text:000000014012D150  00000014    C   rts_unregister_pofx
.text:000000014012D170  0000002F    C   %I64d (%d) %s : ===><===
PoFxUnregisterDevice\n
.text:000000014012D1A0  00000020    C   rts_pnp_fdo_start_dev_delaywork
.text:000000014012D1C0  0000002E    C   %I64d (%d) %s : rts_create_childen_pdos
fail\n
.text:000000014012D1F0  00000039    C   %I64d (%d) %s : Start after stop,so no
need create pdos\n
.text:000000014012D230  00000039    C   %I64d (%d) %s :
IoSetDeviceInterfaceState:enable:failed\n
.text:000000014012D270  00000039    C   %I64d (%d) %s : ===><===
PoFxStartDevicePowerManagement\n

//以下是PNP回调的函数体实现，rts_pnp_fdo_xxx

//对应PNP: IRP_MN_START_DEVICE
.text:000000014012D3E0  00000016    C   rts_pnp_fdo_start_dev
.text:000000014012D400  0000001C    C   rts_pnp_fdo_cancel_stop_dev
.text:000000014012D420  0000002F    C   %I64d (%d) %s : Cancel stop after query
stop \n
```

```
.text:000000014012D450  00000073    C   %I64d (%d) %s : spurious cancel-stop
without query stop first,we still pass it down,Irp->IoStatus.Status is 0x%x \n

//对应PNP: IRP_MN_WAIT_WAKE?
.text:000000014012D4D0  00000019    C   rts_pnp_wait_d0_complete
.text:000000014012D4F0  0000003F    C   %I64d (%d) %s : 0 == fdx-
>CancelSSIsCalling, return directly \n
.text:000000014012D530  00000037    C   %I64d (%d) %s : Wait cancel ss finished
for the %ith \n

//对应PNP: IRP_MN_STOP_DEVICE
.text:000000014012D570  00000015    C   rts_pnp_fdo_stop_dev
.text:000000014012D590  00000044    C   %I64d (%d) %s : KeWaitForSingleObject
NoSSDpcWorkItemPendingEvent \n
.text:000000014012D5E0  0000003B    C   %I64d (%d) %s :
IoSetDeviceInterfaceState::disable:failed\n

.... //略
```

Import页面查看导入的符号，都是WDM （NTOS kernel）的符号链接：

```
Address Ordinal Name    Library

cng
0000000140138010        BCryptSetProperty   cng
0000000140138018        BCryptCloseAlgorithmProvider    cng
0000000140138020        BCryptGenerateSymmetricKey  cng
0000000140138028        BCryptGenerateKeyPair   cng
0000000140138030        BCryptEncrypt   cng
0000000140138038        BCryptExportKey cng
0000000140138040        BCryptGetProperty   cng
0000000140138048        BCryptFinalizeKeyPair   cng
0000000140138050        BCryptDestroyKey    cng
0000000140138058        BCryptDestroySecret cng
0000000140138060        BCryptSecretAgreement   cng
0000000140138068        BCryptDeriveKey cng
0000000140138070        BCryptGenRandom cng
0000000140138078        BCryptImportKeyPair cng
0000000140138080        BCryptOpenAlgorithmProvider cng

ntoskrnl
0000000140138090        KeReadStateEvent    ntoskrnl
0000000140138098        KeReleaseSemaphore  ntoskrnl
00000001401380A0        KeWaitForMultipleObjects    ntoskrnl
00000001401380A8        KeWaitForSingleObject   ntoskrnl
00000001401380B0        ExAllocatePoolWithTag   ntoskrnl
00000001401380B8        ExRaiseStatus   ntoskrnl
00000001401380C0        ProbeForWrite   ntoskrnl
00000001401380C8        MmProbeAndLockPages ntoskrnl
00000001401380D0        MmMapLockedPagesSpecifyCache    ntoskrnl
00000001401380D8        IoAllocateMdl   ntoskrnl
00000001401380E0        IofCallDriver   ntoskrnl
00000001401380E8        IofCompleteRequest  ntoskrnl
00000001401380F0        IoFreeMdl   ntoskrnl
00000001401380F8        IoIs32bitProcess    ntoskrnl
0000000140138100        IoCsqInsertIrp  ntoskrnl
0000000140138108        IoCsqRemoveNextIrp  ntoskrnl
```

```
0000000140138110        __C_specific_handler    ntoskrnl
0000000140138118        wcscat_s    ntoskrnl
0000000140138120        wcscpy_s    ntoskrnl
0000000140138128        wcsncpy_s   ntoskrnl
0000000140138130        RtlInitAnsiString   ntoskrnl
0000000140138138        RtlInitUnicodeString    ntoskrnl
0000000140138140        RtlQueryRegistryValues  ntoskrnl
0000000140138148        MmGetSystemRoutineAddress   ntoskrnl
0000000140138150        RtlAnsiStringToUnicodeString    ntoskrnl
0000000140138158        RtlFreeUnicodeString    ntoskrnl
0000000140138160        RtlCompareMemory    ntoskrnl
0000000140138168        KeInsertQueueDpc    ntoskrnl
0000000140138170        KeSetEvent  ntoskrnl
0000000140138178        KeDelayExecutionThread  ntoskrnl
0000000140138180        KeAcquireSpinLockRaiseToDpc ntoskrnl
0000000140138188        KeReleaseSpinLock   ntoskrnl
0000000140138190        ExFreePoolWithTag   ntoskrnl
0000000140138198        MmBuildMdlForNonPagedPool   ntoskrnl
00000001401381A0        IoDeleteDevice  ntoskrnl
00000001401381A8        IoAllocateWorkItem  ntoskrnl
00000001401381B0        IoFreeWorkItem  ntoskrnl
00000001401381B8        IoQueueWorkItem ntoskrnl
00000001401381C0        IoInvalidateDeviceRelations ntoskrnl
00000001401381C8        IoOpenDeviceRegistryKey ntoskrnl
00000001401381D0        ObReferenceObjectByHandle   ntoskrnl
00000001401381D8        ObfDereferenceObject    ntoskrnl
00000001401381E0        ZwCreateFile    ntoskrnl
00000001401381E8        ZwClose ntoskrnl
00000001401381F0        ZwCreateKey ntoskrnl
00000001401381F8        ZwOpenKey   ntoskrnl
0000000140138200        ZwDeleteKey ntoskrnl
0000000140138208        ZwEnumerateKey  ntoskrnl
0000000140138210        ZwFlushKey  ntoskrnl
0000000140138218        ZwQueryKey  ntoskrnl
0000000140138220        ZwQueryValueKey ntoskrnl
0000000140138228        ZwSetValueKey   ntoskrnl
0000000140138230        ZwPowerInformation  ntoskrnl
0000000140138238        ObQueryNameString   ntoskrnl
0000000140138240        swprintf_s  ntoskrnl
0000000140138248        strncpy_s   ntoskrnl
0000000140138250        DbgPrint    ntoskrnl
0000000140138258        PsGetCurrentThreadId    ntoskrnl
0000000140138260        KfRaiseIrql ntoskrnl
0000000140138268        IoBuildPartialMdl   ntoskrnl
0000000140138270        RtlGetVersion   ntoskrnl
0000000140138278        RtlIsNtDdiVersionAvailable  ntoskrnl
0000000140138280        KeInitializeDpc ntoskrnl
0000000140138288        KeInitializeEvent   ntoskrnl
0000000140138290        KeInitializeSemaphore   ntoskrnl
0000000140138298        KeInitializeTimerEx ntoskrnl
00000001401382A0        IoAttachDeviceToDeviceStack ntoskrnl
00000001401382A8        IoDetachDevice  ntoskrnl
00000001401382B0        IoRegisterShutdownNotification  ntoskrnl
00000001401382B8        IoCsqInitialize ntoskrnl
00000001401382C0        IoRegisterDeviceInterface   ntoskrnl
00000001401382C8        ExFreePool  ntoskrnl
00000001401382D0        MmMapIoSpace    ntoskrnl
00000001401382D8        MmUnmapIoSpace  ntoskrnl
```

```
00000001401382E0        ZwEnumerateValueKey ntoskrnl
00000001401382E8        KeCancelTimer   ntoskrnl
00000001401382F0        IoBuildDeviceIoControlRequest   ntoskrnl
00000001401382F8        IoDisconnectInterrupt   ntoskrnl
0000000140138300        IoGetAttachedDeviceReference    ntoskrnl
0000000140138308        IoUnregisterShutdownNotification    ntoskrnl
0000000140138310        IoSetDeviceInterfaceState   ntoskrnl
0000000140138318        PoRequestPowerIrp   ntoskrnl
0000000140138320        PoSetPowerState ntoskrnl
0000000140138328        ObfReferenceObject  ntoskrnl
0000000140138330        ExUuidCreate    ntoskrnl
0000000140138338        KeSetTimerEx    ntoskrnl
0000000140138340        IoCancelIrp ntoskrnl
0000000140138348        PoCallDriver    ntoskrnl
0000000140138350        PoStartNextPowerIrp ntoskrnl
0000000140138358        PsCreateSystemThread    ntoskrnl
0000000140138360        PsTerminateSystemThread ntoskrnl
0000000140138368        KeAcquireSpinLockAtDpcLevel ntoskrnl
0000000140138370        KeReleaseSpinLockFromDpcLevel   ntoskrnl
0000000140138378        MmUnlockPages   ntoskrnl
0000000140138380        MmAllocateContiguousMemory  ntoskrnl
0000000140138388        MmFreeContiguousMemory  ntoskrnl
0000000140138390        IoAllocateIrp   ntoskrnl
0000000140138398        IoBuildSynchronousFsdRequest    ntoskrnl
00000001401383A0        IoConnectInterrupt  ntoskrnl
00000001401383A8        IoFreeIrp   ntoskrnl
00000001401383B0        IoGetDmaAdapter ntoskrnl
00000001401383B8        IoGetDeviceProperty ntoskrnl
00000001401383C0        MmGetPhysicalAddress    ntoskrnl
00000001401383C8        RtlUnicodeToMultiByteN  ntoskrnl
00000001401383D0        KeClearEvent    ntoskrnl
00000001401383D8        KeQueryActiveProcessors ntoskrnl
00000001401383E0        KeBugCheckEx    ntoskrnl
00000001401383E8        ZwSetSecurityObject ntoskrnl
00000001401383F0        IoDeviceObjectType  ntoskrnl
00000001401383F8        IoCreateDevice  ntoskrnl
0000000140138400        ObOpenObjectByPointer   ntoskrnl
0000000140138408        RtlGetDaclSecurityDescriptor    ntoskrnl
0000000140138410        RtlGetGroupSecurityDescriptor   ntoskrnl
0000000140138418        RtlGetOwnerSecurityDescriptor   ntoskrnl
0000000140138420        RtlGetSaclSecurityDescriptor    ntoskrnl
0000000140138428        SeCaptureSecurityDescriptor ntoskrnl
0000000140138430        _snwprintf  ntoskrnl
0000000140138438        RtlLengthSecurityDescriptor ntoskrnl
0000000140138440        SeExports   ntoskrnl
0000000140138448        RtlCreateSecurityDescriptor ntoskrnl
0000000140138450        _wcsnicmp   ntoskrnl
0000000140138458        wcschr  ntoskrnl
0000000140138460        RtlAbsoluteToSelfRelativeSD ntoskrnl
0000000140138468        RtlAddAccessAllowedAce  ntoskrnl
0000000140138470        RtlLengthSid    ntoskrnl
0000000140138478        IoIsWdmVersionAvailable ntoskrnl
0000000140138480        RtlSetDaclSecurityDescriptor    ntoskrnl
0000000140138488        ExAllocatePoolWithQuotaTag  ntoskrnl
0000000140138490        PsGetVersion    ntoskrnl
0000000140138498        ZwQuerySystemInformation    ntoskrnl
00000001401384A0        KeLowerIrql ntoskrnl
```

```
HAL
0000000140138000       KeStallExecutionProcessor   HAL
```

结论：根据.sys的分析，此驱动是纯WDM实现的。因为内核接口全部调用WDM/NT kernel接口， 完全没有调用微软的miniport框架例如Storport, SDHC，SDBUS框架封装后的接口。

但微软已不推荐WDM驱动开发，参考：Introduction to WDM 。新驱动优先使用KMDF框架。

3. 参考文章

https://voidsec.com/windows-drivers-reverse-engineering-methodology/#remote-kernel-debugging

https://blog.csdn.net/qq_24481913/article/details/131643283

https://learn.microsoft.com/en-us/windows-hardware/drivers/kernel/